Auddard & Co.

Arithmetic Expression Evaluator in C++

User's Manual

Version 1.3

Revision History

Date	Version	Description	Author
28/Nov./23	1.0	Wrote Purpose and Introduction together, skipped Getting Started to be done the last. Filled out the Glossary as necessary	Audrey Pan, Sriya Annem, Morgan Nguyen, Lauren D'Souza, Ella Nguyen, Tanu Sakaray
30/Nov./23	1.1	Covered Advanced Features and Troubleshooting. Filled out the Glossary as we felt necessary	Audrey Pan, Sriya Annem, Morgan Nguyen, Lauren D'Souza, Ella Nguyen, Tanu Sakaray
2/Dec./23	1.2	Did the Examples and FAQ and then returned back to the beginning to do the Getting Started Section.Added to the Glossary as necessary	Audrey Pan, Sriya Annem, Morgan Nguyen, Lauren D'Souza, Ella Nguyen, Tanu Sakaray

Table of Contents

Test Case

1. Purpose

The purpose of this C++ calculator program is to offer users a versatile and efficient tool for performing mathematical computations following the principles of PEMDAS (Parentheses, Exponents, Multiplication and Division, Addition and Subtraction). This software aims to simplify complex mathematical tasks by providing a user-friendly interface that accepts input expressions and produces accurate results. With a commitment to adhering to the rules of mathematics, the calculator ensures reliability and precision in its calculations. Whether users are students grappling with algebraic equations or professionals needing quick numeric solutions, this calculator serves as a reliable companion, fostering a seamless and intuitive mathematical experience.

2. Introduction

Welcome to our C++ Calculator, a powerful tool designed to streamline your mathematical computations with precision and ease. In today's fast-paced world, where efficiency is key, this calculator stands out as a valuable resource for users at all levels of mathematical proficiency. From basic arithmetic to more complex algebraic expressions, this software empowers users to handle diverse mathematical challenges effortlessly. In this guide, we will walk you through the purpose and functionality of the calculator, ensuring that you harness its full potential. But before we delve into the advanced features and troubleshooting aspects, let's start with the basics – understanding the purpose and getting started with the installation and execution process. To make your experience even smoother, we've made our **source code** accessible on GitHub, enabling you to stay up-to-date with the latest enhancements and contribute to the calculator's continuous improvement.

Within this comprehensive user manual, we've organized the content to guide you through every facet of our C++ Calculator, ensuring a seamless and enriching user experience. The "Getting Started" section presents a step-by-step guide, detailing the entry of expressions, the utilization of various operators and functions, and the interpretation of results. As you delve into the "Advanced Features," discover the software's capabilities, such as saving and loading expressions, and defining custom variables and functions. For users encountering challenges, the "Troubleshooting" section offers a structured approach to address issues, providing clear solutions for common problems like invalid expressions or incorrect outputs. Our "Examples" section demonstrates the versatility of the calculator, showcasing how it excels in evaluating different types of arithmetic expressions. To enhance your understanding, we've included a "Glossary of Terms" defining technical language used throughout the manual, ensuring clarity in communication. Lastly, our "FAQ" section anticipates and addresses your queries, offering concise introductions to key topics and providing the information you need for a confident and proficient interaction with our C++ Calculator.

3. Getting started

Using the Arithmetic Expression Evaluator

Operators:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Exponentiation (^)
- Parentheses ( )

Order of operations: The evaluator follows the standard order of operations (PEMDAS)

- Parentheses > Exponents > Multiplication/Division/Modulus > Addition/Subtraction.

Starting the Program:

- Open up our Git link provided
- There will be a green button labeled "Code"
- Click onto the button and click "Download ZIP"
- Open the .zip file labeled: "EECS348-arithmetic-calculator-main"
- Open up terminal and navigate to wherever the "EECS348-arithmetic-calculator-main" folder was downloaded or saved to
- Once you are in the "EECS348-arithmetic-calculator-main" folder, navigate to the folder named "calculator"
- once in the folder, enter "make" in the terminal to make the executable
- Aftwards, to run the program, enter "./arith_calc"

Entering Expressions:

- Input: Begin by entering your arithmetic expression using the supported operators:
- Be sure to separate operators with a space
  - Ex: 3+4 → 3_+_4
  - Ex. (3^3)/(5+3) → (3_^_3)_/_(5_+_3)
    - NOTE: THE UNDERSCORES REPRESENT THE SPACES - DO NOT ENTER UNDERSCORES INTO THE PROGRAM, this is just for understanding
- Unary operators do not need a space
  - Ex. -1 → -1
- Parentheses: Use parentheses to explicitly define the order of operations.
  - Ex: (2 + 3) * 4 ensures addition before multiplication.

Interpreting Results:

- Results Display: After entering the expression, the calculator displays the result of the evaluation.
- Handling Errors: If an error occurs (e.g., division by zero), the calculator provides an error message explaining the issue.

Example Expressions:

- Basic Calculation: 2 + 3 * 4 will result in 14, following the order of operations.
- Use of Parentheses: (2 + 3) * 4 will yield 20 as parentheses take precedence.
- Exponentiation: 2^3 represents 2 raised to the power of 3, resulting in 8.

Tips for Efficient Use:

- Clear Input: Ensure the expression is clear and follows the correct syntax with appropriate operator usage.
  - Ex. of Invalid Input: * 5 + 2 (the * operator lacks operands on the left, making the expression invalid

Grouping Operations: Use parentheses to clarify the sequence of operations if needed.

Handling Errors: Pay attention to error messages to identify and rectify any issues in the expressions entered.

Decimal Numbers: For decimal numbers, use the dot notation (e.g., 3.14) to represent fractional values.

4. Advanced features

While not many advanced features have been added yet, this remains a possibility for future updates to the calculator. Currently, our calculator will continue to prompt you to enter an expression until q is entered or until an invalid expression is entered. On a similar note, if an invalid expression is entered, the program will provide an error message describing why it is invalid.

Custom variables are not currently able to be created through command lines, however one can create custom variables directly in the **source code** of the program. This can be done by creating string variables with the custom values you want stored, and then call “evaluator.evaluate(expression), where expression is a string variable of the expression containing the custom variable. This code should go before the following line in the **main function**:

```
cout << "Enter arithmetic expression or type q to quit: "; //prints out the user input prompt
```

For example, to calculate the area of a circle, one could use the following code:

```
string radius = “4”; //create a variable for the radius
string pi = “3.14”;  //create a variable for pi
string custom_expression =  pi + “ * ” + radius + “^2”; //expression that represents 3.14 * 4^2 (area of a
circle with radius 4)
double result = evaluator.evaluate(custom_expression); //stores the evaluated result into a variable called
result
cout << “Custom expression result: ” << result << endl; //prints the result to the screen
```

We hope to add custom variable functionality through the command line in future updates.

5. Troubleshooting

Issue	**Potential Solutions**
Program Deemed Expression as Invalid	● Read the error message, the program should describe the error ● Check to make sure there is an even amount of operands and operators ● Ensure that all parentheses are closed ● Use specific operators: + for addition, - for subtraction, * for multiplication, / for division, ^ for exponentiation, and % as a modulo operator. ● Do not divide by zero.
Program Gave Incorrect Output/Result	● Check to make sure that unary operators are directly next to operands, and that arithmetic operators have a space in between ● The program only accepts certain operators: Use + for addition, - for subtraction, * for multiplication, / for division, ^ for exponentiation, and % for modulo.
Leaving the Program	● While the program is optimized to run with little issues, if the program seems to buffer, give it just a moment to catch up. ● Use the “Quit” feature by entering “q” as an input ● Kill the program by using CTRL + C

6. Examples

- General

  - Begin running the software through the terminal. Copy and paste the expression or type in each character.
  - No "=" needed at the end of the expression; the program assumes that the expression needs to be evaluated
  - Dividing by zero will result in an error
  - Any symbols besides digits 0-9, decimals, and other operators will not be recognized and will result in an error
- Expressions
  - Any PEMDAS operations are valid
  - Extraneous parentheses and spaces will be handled appropriately
  - Use the "^" symbol to notate exponents
  - Use the "/" symbol to notate division
  - Use the "%" symbol to notate modulo operators
  - To notate square roots, take your number to the power of (½)
    - For example, 2 ^ (1 / 2)

7. Glossary of terms

- **Arithmetic Expression:** combination of operators, numeric values, and sometimes parentheses
- **Arithmetic Operator:** Perform addition, subtraction, multiplication, division, exponentiation, and modulus operations
- **Operand:** the object or quantity that is operated on
- **Modulo Operator:** Gives the remainder of a division
- **Unary Operator:** An operator that takes a single operand in an expression or a statement
- **Source Code:** The .cpp file with the C++ code necessary to run an arithmetic calculator
- **Main function:** A function in the source code where all other functions are called. The header of the function is int main()

8. FAQ

Which operators does the calculator support for arithmetic operations?

The calculator supports basic arithmetic operators: addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (^).

Can I use parentheses in my expressions for grouping?

Yes, parentheses can be used to group parts of the expression to dictate the order of operations.

Does the calculator handle decimal numbers or only integers?

The calculator can handle both integers and decimal numbers.

How does the calculator manage the order of operations (PEMDAS)?

It follows the standard order of operations (PEMDAS/BODMAS), giving precedence to parentheses, exponents, multiplication, division, addition, and subtraction.

Are there limitations or restrictions on the length of the expression?

The calculator typically does not have a set limit on expression length but might be subject to practical computational constraints.

Can I include spaces in my expression for readability?

Yes, spaces can be added in the expression for readability without affecting the evaluation.

What happens if I forget to close a parenthesis in my expression?

The calculator will display an error message indicating unmatched parentheses.

Does the calculator provide clear error messages for incorrect expressions?

Yes, it provides informative error messages for various issues encountered during expression evaluation.

Can I enter multiple expressions in a single session, or do I need to rerun the program for each one?

You can enter multiple expressions in a single session. The calculator prompts for a new expression after each evaluation. However, if the expression fails then you need to rerun the program.

Does the calculator support scientific notation for large or small numbers?

Currently, the calculator might not directly support scientific notation (e.g., 1.23e4 for 12,300), but you can input large or small numbers directly using standard decimal notation.

How does the calculator handle unary operators like '+', '-'?

Unary operators at the beginning of an expression are supported for positive (+) and negative (-) numbers